REMARKS

I. Introduction

In response to the Office Action dated August 25, 2006, no claims have been canceled, amended or added. Claims 1-18 remain in the application. Re-examination and re-consideration of the application is requested.

II. Claim Objections

In paragraph (2) of the Office Action, claims 6, 12 and 18 were objected to for certain informalities, namely that they are dependent upon the wrong claims.

Applicants' attorney respectfully traverses these objections, and asserts that these claims are dependent upon the proper claims, namely their respective immediately preceding claim.

III. Prior Art Rejections

A. The Office Action Rejections

In paragraphs (3)-(5) of the Office Action, claims 1-18 were rejected under 35 U.S.C. §102(b) as being anticipated by U.S. Patent No. 6,785,848 (Glerum).

Applicants' attorney respectfully traverses these rejections.

B. The Applicants' Independent Claims

Independent claims 1, 7 and 13 are generally directed to providing contextual diagnostic data at a point of failure of a software program. Claim 1, which is representative, is a method for providing contextual diagnostic data at a point of failure of a software program, comprising:

(a) registering callbacks for one or more modules and sub-applications within the program;

(b) examining a call stack for the program upon failure of the program;

(c) notifying the registered callbacks for the modules and sub-applications based on the examined call stack;

(d) performing callback processing, wherein the notified callbacks of the modules and sub-applications extract and supply context data; and

(e) packaging the context data supplied by the notified callbacks of the modules and sub-applications.

C. The Glerum Reference

Glerum describes a method for categorizing information regarding a failure in an application program module. The failure may be a crash, a set-up failure or an assert. For a crash, a name of an executable module where the crash occurred in the application program module, a version number of the executable module, a name of a module containing an instruction causing the crash, a version number of the module and an offset into the module with the crashing instruction are determined. This bucket information is then transmitted to a repository for storage in a database. The database may be examined to determine fixes for the bug that caused the crash.

D. The Applicants' Invention is Patentable Over the Reference

The Applicants' invention, as recited in independent claims 1, 7 and 13, is patentable over the Glerum reference, because it contains limitations not taught by the reference.

Specifically, Glerum does not teach or suggest the specific steps or functions recited in Applicants' claims for providing contextual diagnostic data at a point of failure of a software program.

Nonetheless, the Office Action asserts the following:

> 4. Claims 1-18 are rejected under 35 U.S.C. 102(b) as being anticipated by Glerum et al (US 6,785,848 B1).
>
> 5. Claim 1: Glerum discloses a method for providing contextual diagnostic data at a point of failure of a software program comprising:
>
> a. Registering callbacks for one or more modules and sub-applications within the program (Col 5, line 47-56);
>
> b. Examining a call stack for the program upon failure of the program (Col 6, line 23-45);
>
> c. Notifying the registered callbacks for the modules and sub-applications based on the examined call stack (Col 7, line 30-40);
>
> d. Performing callback processing, wherein the notified callbacks of the modules and sub-applications extract and supply context data (Col 8, line 37-50); and
>
> e. Packing the context data supplied by the notified callbacks of the modules and sub-application (Col 8, line 60-67).

Applicants' attorney disagrees with this analysis.

With regard to the limitations "registering callbacks for one or more modules and sub-applications within the program," the cited locations in Glerum are set forth below:

Glerum: col. 5, line 47-56

The system 200 also comprises an exception filter 220. Exception filters are well-known in the art and may be registered by program modules when the operating system 35 is started. When a failure (an exception) occurs, the exception filter 220 code is executed. For example, suppose a failure occurs while executable program 210 is executing instructions running module 215 at location 225. If executable program 210 has registered exception filter 220 with the operating system, then the exception filter 220 is executed when executable program 210 encounters an exception.

The above portions of Glerum merely describe an exception filter, which is for the type of exception being handled (for unhandled exceptions, there will typically be a single exception filter).

In contrast, Applicants' claimed invention registers callbacks for modules and sub-applications within the program, not for the exception being handled. As described below, these callbacks are then invoked, depending on the location where the crash occurred, while in Glerum the exception handler is called depending on which type of exception occurred.

With regard to the limitations "examining a call stack for the program upon failure of the program," the cited locations in Glerum are set forth below:

Glerum: col. 6, line 23-45

When a failure in the application program module 205 occurs at location 225, the operating system 35 (FIG. 1) throws the application program module 205 out of memory and the exception filter 220 executes the failure reporting executable 230. The failure reporting executable 230 then must determine the type of failure that has occurred and determine how to categorize the failure for transmission to the repository 235. Typically, the type of failure is either a crash, a set-up failure or an assert.

Based upon the type of failure, the failure reporting executable 230 then determines what relevant information to retrieve from the application program module to uniquely identify, i.e. categorize, the failure. In many cases, uniquely identifying the failure means determining the location of the failure. Typically, the categorization, or location information, of the failure is sent to the repository as a bucket. A bucket is a set of information uniquely defining the location of the failure. If a bucket from one failure matches a bucket from another failure, then it is assumed that both failures are caused by the same bug. Although not always accurate (because more than one bug may be at the same location), this assumption that failures with the same bucket information are caused by the same bug allows for effective organization in the repository, as will be further described below.

The above portions of Glerum merely describe a failure occurring, an exception filter executing a failure reporting executable, and the failure reporting executable categorizing the failure by identifying where the failure occurred.

However, nowhere do the above portions of Glerum teach or suggest examining the call stack for the program upon failure of the program, in the context of determining which callbacks to notify, as described below.

With regard to the limitations "notifying the registered callbacks for the modules and sub-applications based on the examined call stack," the cited locations in Glerum are set forth below:

> Glerum: col. 7, line 30-40
>
> After the failure is categorized by determining bucket information and the bucket information is sent to the repository, the repository determines whether the bucket information of the failure matches the bucket information of any previously reported failures. This determination is helpful to determine whether the failure is a new failure (i.e., one that has not been reported before), to determine whether there is a fix for the failure, to determine whether the software developers have requested that more information be collected regarding this type of failure, etc.

The above portions of Glerum merely describe determining bucket information, and then sending the bucket information to a repository.

However, nowhere do the above portions of Glerum teach or suggest notifying the registered callbacks for the modules and sub-applications based on the examined call stack. There are no callbacks in Glerum. Moreover, as described below, these callbacks are invoked in Applicants' claimed invention, depending on where the crash occurs, to supply contextual data for the crash.

With regard to the limitations "performing callback processing, wherein the notified callbacks of the modules and sub-applications extract and supply context data," the cited locations in Glerum are set forth below:

> Glerum: col. 8, line 37-50
>
> As part of requesting and receiving the additional data, the failure reporting executable may generate a minidump. The minidump is essentially a collection of relevant information that provides an autopsy of the crashed application program module. The minidump may comprise information about the state of the process at the time the dump was collected that is helpful in understanding what caused the failure. Typically, the minidump comprises brief information about the computer (such as operating system and CPU); a list of all the threads in the process with their CONTEXTS (an operating system term describing the current state of the thread's execution on the CPU) and memory stack; a list of all modules loaded in the process and their relevant information (name, version number, where they are loaded into the process space, etc.); and the global data associated with specific modules (such as mso.dll, outllib.dll and the module containing the failure).

The above portions of Glerum merely describe minidumps generated by the failure reporting executable.

However, the above portions of Glerum do not teach or suggest performing callback processing, wherein the notified callbacks of the modules and sub-applications extract and supply context data. As noted above, there are no callbacks in Glerum.

With regard to the limitations "packaging the context data supplied by the notified callbacks of the modules and sub-applications," the cited locations in Glerum are set forth below:

> Glerum: col. 8, line 60-67
>
> In a preferred embodiment, the data is sent to the repository as .cab files or in another compressed format. .cab is used as a file extension for cabinet files which are well-known. Cabinet files are multiple files compressed into one and extractable with the extract.exe utility.
>
> It should be understood that the data sent to the repository may be used by developers to diagnose the failure and, if possible, develop a fix for the failure.

The above portions of Glerum merely describe the minidump being packaged as a cabinet file.

However, the above portions of Glerum do not teach or suggest packaging the context data supplied by the notified callbacks. Again, there are no callbacks in Glerum.

Thus, Glerum does not anticipate or render obvious Applicants' claimed invention. Moreover, the various elements of Applicants' claimed invention together provide operational advantages over Glerum. In addition, Applicants' claimed invention solves problems not recognized by Glerum.

Specifically, there are some very basic differences between Glerum and Applicants' claimed invention.

Applicants' claimed invention is a client-side determination of what diagnostic data to send to the vendor, i.e., dynamically (programmatically) determined by the client at the point of failure. Glerum's strategy for requesting additional data is established a priori on the server-side, and no new contextual diagnostic data is generated at the point of failure.

Applicants' claimed invention extracts run-time context data supplied by callbacks registered by modules or sub-applications, where the callbacks are notified based on examination of the call stack at the time of failure of the program. Glerum, on the other hand, merely describes typical exception handling by exception type, along with minidumps, bucketing and typical error reporting.

The motivation for Applicants' claimed invention results from the experience in trying to find the root cause of a problem while processing customer error reports (CERs). The problem is that, when a minidump or stack dump is received by the vendor, it may not have all the context data necessary for analyzing and solving the problem.

The approach of Applicants' claimed invention is to have modules and sub-applications register "callbacks" that are called when the call stack has addresses that fall in a registered range for those modules and sub-applications. At that point, the specific callback can glean specific context data that can be used as diagnostic data. The point is that, at the time of crash, the client application that has crashed examines the stack and, based on the addresses on the stack, notifies a specific callback to extract and supply context data as diagnostic data.

Thus, Applicants' attorney submits that independent claims 1, 7 and 13 are allowable over Glerum. Further, dependent claims 2-6, 8-12 and 14-18 are submitted to be allowable over Glerum in the same manner, because they are dependent on independent claims 1, 7 and 13, respectively, and thus contain all the limitations of the independent claims. In addition, dependent claims 2-6, 8-12 and 14-18 recite additional novel elements not shown by Glerum.

IV. Conclusion

In view of the above, it is submitted that this application is now in good order for allowance and such allowance is respectfully solicited.

Should the Examiner believe minor matters still remain that can be resolved in a telephone interview, the Examiner is urged to call Applicants' undersigned attorney.

Respectfully submitted,

GATES & COOPER LLP
Attorneys for Applicants

Howard Hughes Center
6701 Center Drive West, Suite 1050
Los Angeles, California 90045
(310) 641-8797

Date: November 27, 2006

By: /George H. Gates/
Name: George H. Gates
Reg. No.: 33,500

GHG/

G&C 30566.315-US-01